Irene Truchado Mazzoli

Pareja PT41

Arquitectura de Ordenadores 2019-2020

Práctica 3 – Memoria

GRUPO 1362

ÍNDICE

EJERCICIO 0

INFORMACIÓN SOBRE LA CACHÉ DEL SISTEMA

Ejecutando el comando **getconf -a |grep -i cache**, obtenemos la siguiente información:

Podemos observar los diferentes niveles de caché que existen en los ordenadores de los laboratorios de la universidad. Identificamos tres niveles de caché:

1. *Caché de nivel 1*

   Lo primero a destacar, es que únicamente existe separación de cachés de datos e instrucciones en este nivel. Ambas tienen un tamaño de 32768 Bytes, es decir, 32 KB y un tamaño de bloque de 64 Bytes, por lo que cada una tiene 512 bloques (32768/64). También observamos que son cachés asociativas de 8 vías. Teniendo esta información, podemos determinar que cada vía accede a 64 bloques (512/8).

2. *Caché de nivel 2*

   Esta caché es asociativa a 4 vías y tiene un tamaño de 262144 Bytes, es decir, 256 KB, los cuales se dividen en bloques de 64 Bytes. Sabiendo esto, calculamos que hay 4096 bloques (262144/64) y que cada vía accede a 1024 bloques (4096/4) de 64 Bytes cada uno.

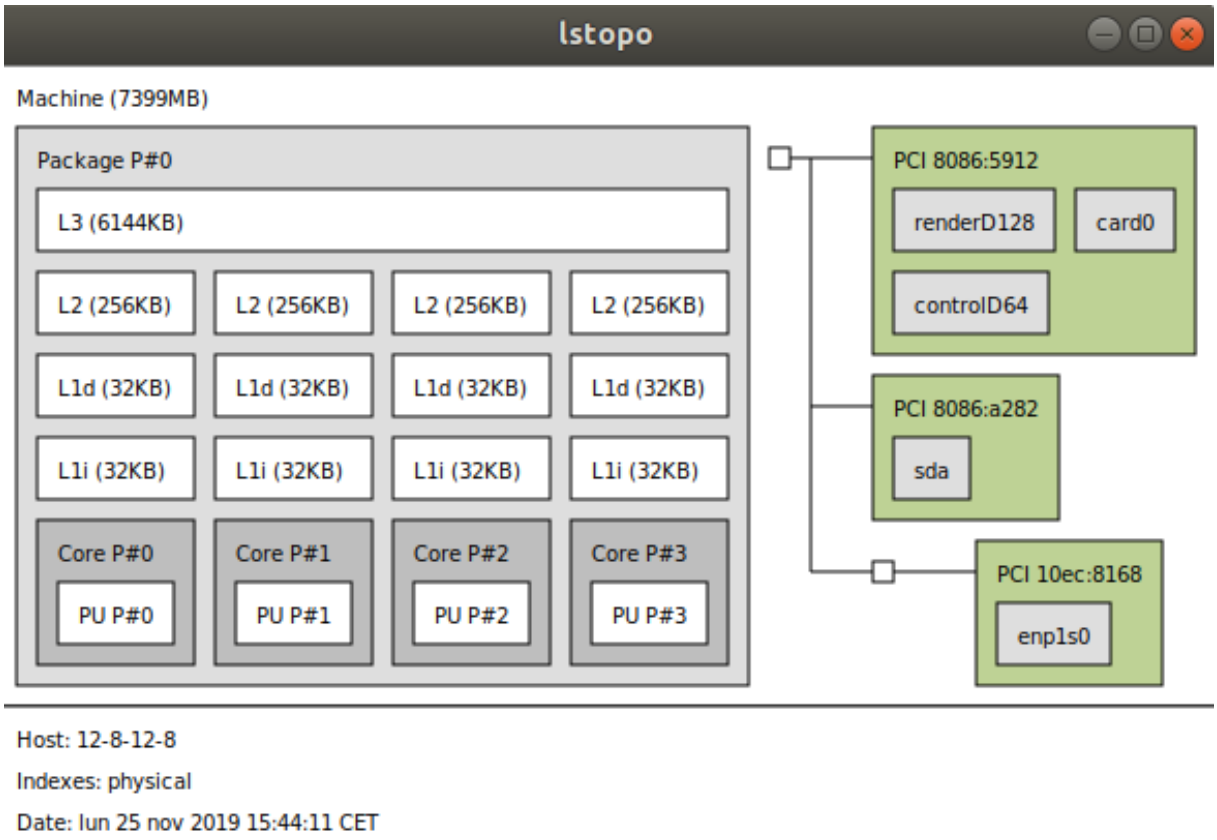
3. *Caché de nivel 3*

   Por último, esta caché es asociativa a 12 vías y tiene un tamaño de 6291456 Bytes, es decir, 6144 KB, los cuales se dividen en bloques de 64 Bytes. Sabiendo esto, calculamos que hay 98304 bloques (6291456 /64) y que cada vía accede a 8192 bloques (98304/12) de 64 Bytes cada uno.

También visualizamos una caché de nivel 4, pero esta no tiene un tamaño definido, por lo que se entiende que el equipo no la utiliza.

Podemos concluir que la memoria caché de los ordenadores de los laboratorios de la universidad utiliza una jerarquía de memoria de tres niveles, de tipo asociativo y varias vías. A medida que se desciende a niveles mayores de caché, su tamaño aumenta, pero probablemente el tiempo de acceso a ellas también lo haga.

Ejecutando el comando **lstopo**, obtenemos información adicional de nuestro equipo.



Efectivamente, comprobamos que la estructura jerárquica organizada en tres niveles que habíamos individuado antes era correcta, así como la separación de caché y datos únicamente en el nivel 1. También observamos que los tamaños de cada memoria coinciden con los vistos anteriormente.

Si nos fijamos en la caché de primer nivel, observamos que indica “PU P#”. Esto significa “Processing Unit Processor”, es decir, los elementos de procesamiento dentro de los núcleos de la CPU. En nuestro caso, tenemos 4 cores y cada uno de ellos tiene un elemento de procesamiento.

EJERCICIO 1

MEMORIA CACHÉ Y RENDIMIENTO

En este ejercicio, se pretende evaluar cómo el patrón de acceso a los datos puede mejorar el aprovechamiento de las memorias caché del sistema.

En primer lugar, se ha calculado el valor de P que se utiliza en todos los ejercicios.

$$P = (num\_pareja \bmod 7) + 4$$

$$\text{pareja PT41: } P = (41 \bmod 7) + 4 = 6 + 4 = 10$$

$$\mathbf{P = 10}$$

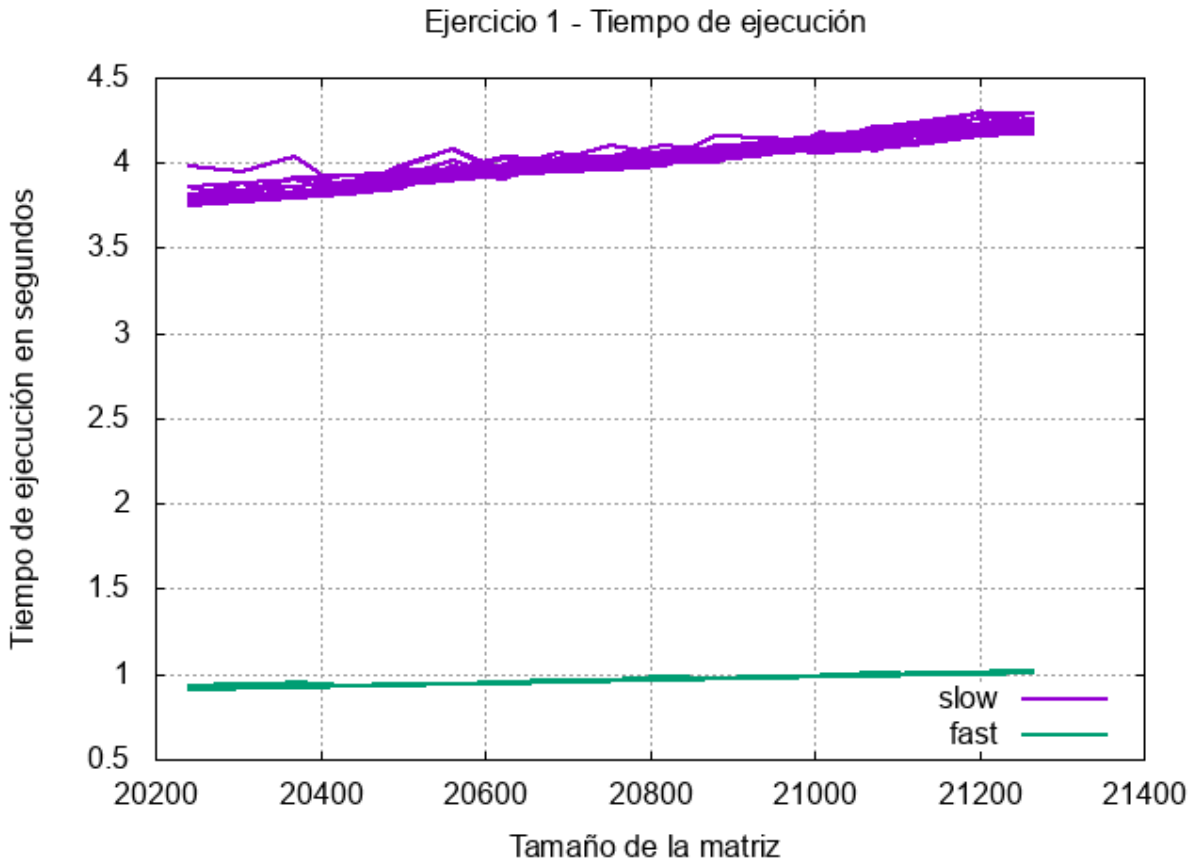
Para este ejercicio, se ha tomado como base el script entregado junto al material de la práctica y a partir de ahí se han ido incluyendo funcionalidades.

En primer lugar, se ha añadido la posibilidad de introducir el número de pruebas a repetir por teclado, para que no sea algo fijo si no que se pueda ir modificando cada vez. Tras varias repeticiones y tras obtener gráficas con picos muy pronunciados, se ha llegado a la conclusión que 10 pruebas es un número suficiente para que los resultados sean homogéneos y así lo refleje la gráfica.

Por otro lado, dentro del bucle for se han ido ejecutando de forma intercalada los programas *slow* y *fast*. Por cada repetición, se añade una nueva fila al fichero *slow_fast_time.dat* que contiene el tamaño de la matriz, el tiempo que ha tardado *slow* y el tiempo que ha tardado *fast*. Al ejecutar los programas de este modo, nos aseguramos de que en cada ejecución los datos que necesita la memoria caché no sean exactamente los mismos que los utilizados anteriormente.

Finalmente, se ha calculado la media y guardado en un fichero temporal. Esto se hace porque el tiempo de ejecución de un mismo programa varía, y al realizar la media de estos intentos lo que hacemos es minimizar componentes aleatorias que no interesan.

Por último, el script genera la siguiente gráfica con los resultados, utilizando GNUPlot.



Slow_fast_time.png

La gráfica obtenida resulta satisfactoria, ya que no presenta picos pronunciados y la diferencia entre el tiempo de ejecución de *slow* y de *fast* es evidente.

¿Por qué el programa *fast* es más rápido? El motivo está en que, a diferencia de *slow*, este suma las entradas de las matrices por filas. Esto es más eficiente siempre, porque al acceder por filas estas se cargan por completo en memoria de una sola vez, mientras que las columnas se van cargando fila por fila. De hecho, cuanto mayor es la matriz más se acentúa esto, ya que más filas se tienen que cargar en memoria.

Además de este motivo, el tiempo es mucho mayor para matrices de mayor tamaño debido al bucle anidado, que incrementa cuadráticamente el tiempo de ejecución si aumenta el tamaño de la matriz.

EJERCICIO 2

TAMAÑO DE LA CACHÉ Y RENDIMIENTO

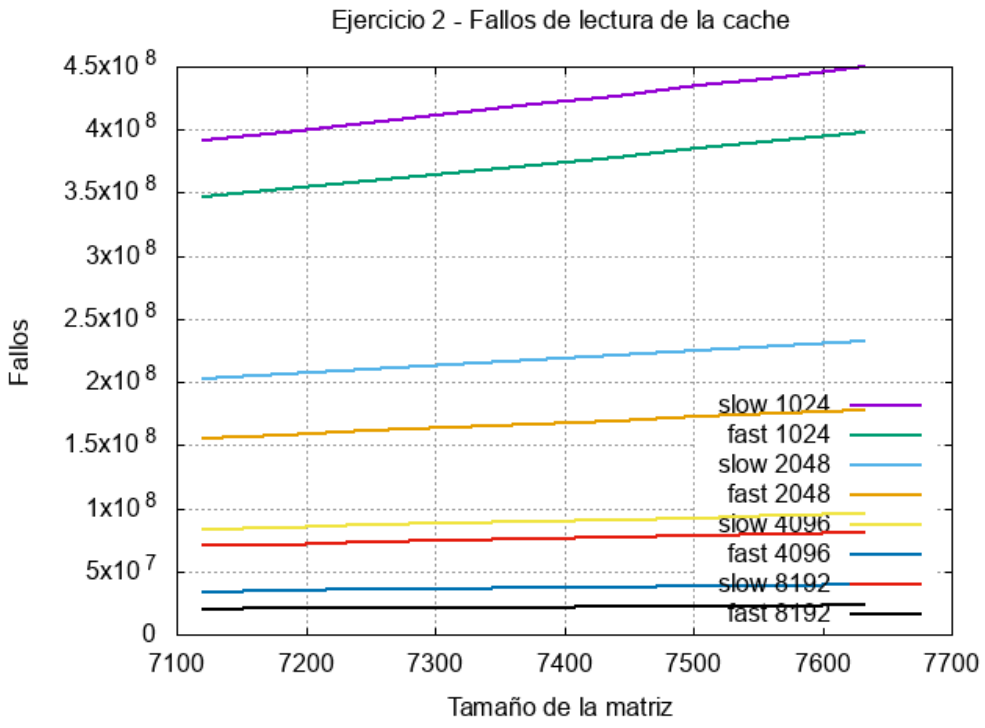
El objetivo de este ejercicio es el de evaluar cómo la configuración de las memorias caché puede afectar al rendimiento de un programa.

Para ello, se han tomado datos de la cantidad de fallos caché producidos en la lectura y escritura al utilizar *slow* y *fast* con diferentes tamaños de matriz y de caché L1. Dicha caché se ajusta mediante *Valgrind*, mediante la utilidad *cachegrid*.

En este caso, no hemos realizado varias repeticiones, si no solo una, ya que la herramienta *Valgrind* ya se encarga de que en la simulación se obtengan siempre los mismos resultados.

El programa **ej2.sh** funciona de la siguiente manera. Mediante un bucle, se van recorriendo los tamaños de matriz variando entre 2000+512*P y 2000+512*(P+1) y con un incremento en saltos de 64 unidades. Para cada uno de estos tamaños, tenemos otro bucle que recorre, esta vez, los tamaños de caché de primer nivel de 1024, 2048, 4096 y 8192 Bytes. Ahora bien, para cada uno de estos tamaños se ejecuta la función *cachegrid* de *Valgrind*, para un tamaño de memoria de 8388608 Bytes (8 MB), una única vía y un tamaño de línea de 64 Bytes. Esto se realiza de manera intercalada para el *slow* y para el *fast*, y se van guardando los resultados en ficheros denominados *cache_<tamCache>.dat*.

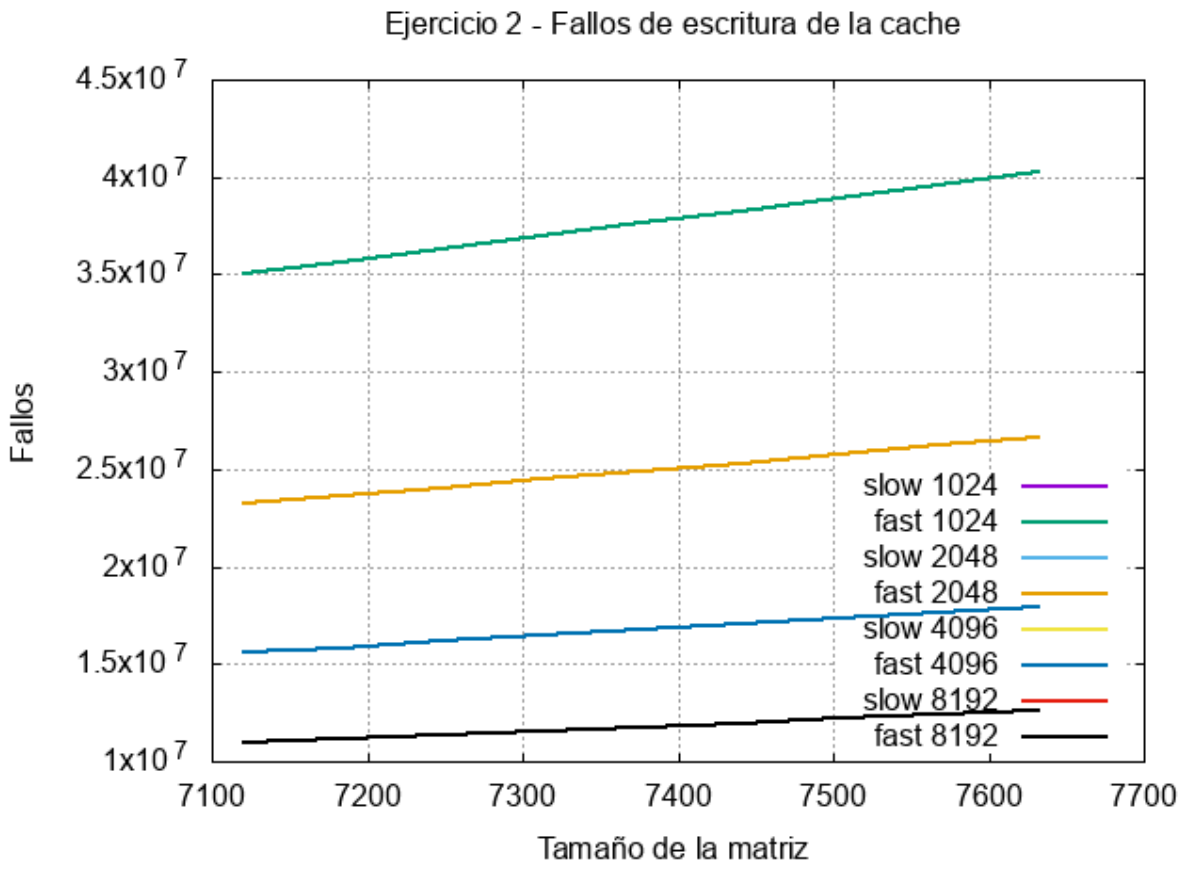
Finalmente, el script genera las siguientes gráficas con los resultados, utilizando GNUPlot.



cache_lectura.png

En esta gráfica, se obtienen los resultados esperados, es decir, líneas sin picos marcados y una notable diferencia entre los fallos de lectura en cachés de tamaño pequeño (más fallos) y las de mayor tamaño (menos fallos). Esto se debe a que caben más bloques simultáneamente en la memoria cuanto más grande sea esta, por lo que el reemplazo y, por lo tanto, el fallo va a ser menor.

En general, si fijamos un tamaño en la caché, el número de fallos aumenta junto con el tamaño de las matrices, ya sea lectura, escritura, *slow* o *fast*. Además, también notamos que *slow* presenta un mayor número de fallos en cualquier tamaño, debido al motivo explicado anteriormente. Cuando *fast* carga un bloque en memoria, los próximos datos que requiera van a estar en ese mismo bloque, y no va a tener que cargar otro hasta que no lo haya leído entero. Sin embargo, *slow* tiene que acceder a un bloque distinto en cada lectura de una misma columna, por lo que siempre va a ser menos eficiente. Por cada elemento de una misma columna que recorra, siempre se va a producir un fallo.



cache_escritura.png

En esta gráfica, notamos que el resultado es diferente al anterior. Puede parecer que “faltan” líneas, pero esto se debe a que en realidad las líneas de *slow* y de fast están coincidiendo para cada tamaño, por lo que sólo se muestran 4 en vez de 8. Esto es debido a que ambos escriben una vez por cada elemento de la matriz.

EJERCICIO 3

CACHÉ Y MULTIPLICACIÓN DE MATRICES

La primera parte de este ejercicio consiste en desarrollar dos programas (*mult.c* y *mult_trans.c*) que multipliquen matrices normales y traspuestas. Se ha comprobado que reportan la misma matriz resultante resultado de ambas es el mismo.

Posteriormente, se ha realizado un script **ej3.sh** en el que se ha desarrollado la funcionalidad necesaria para estudiar los resultados de rendimiento y los fallos de lectura y escritura en caché. Este programa realiza varias pruebas para varios tamaños de matrices al ejecutar las dos versiones, intercaladas, de la multiplicación de matrices, calculando tanto los tiempos de ejecución como los fallos de la caché.

Este ejercicio se ha intentado ejecutar tanto en los laboratorios de la universidad como en mi ordenador personal, pero el tiempo de ejecución era demasiado elevado, incluso disminuyendo el número de repeticiones y aumentando el número de paso, por lo que no se han obtenido resultados satisfactorios.